Table of Contents

ELEC 4700 Assignment 1 W2019 Baldeep Kooner 101004107

Part 1: Electron Modelling

The following equation was used to calculate the thermal velocity using T = 300 K.

$$\overline{v^2} = \frac{2kT}{m}$$

```
m0 = 9.10938215e-31; %rest mass of an electron (kg)
mn = 0.26*m0; %effective mass of an electron (kg)
tau = 0.2e-12; %mean time between collisions (s)
Kb = 1.38064852e-23; %boltzmann constant (J/K)
T = 300; %temperature (K)
vth = sqrt(2*Kb*T/mn) %thermal velocity
```

```
vth =

   1.8702e+05
```

The mean free path, l, was calculated as:

```
l = vth*tau
```

```
l =

   3.7404e-08
```

The following variables are used to set the limits of the simulations and help model the desired electron behaviour

```
nElectrons = 10000;
nTime = 1000;
time = zeros(1, nTime);
dt = (100e-9)/vth/100;
temperature = zeros(1, nTime);
right = 0;
left = 0;

x = linspace(0, 200, 400)*10^(-9);
```

```
y = linspace(0, 100, 400)*10^(-9);
```

The following arrays represent the electrons' positions and velocities they must be initialized. The initial position of each electron in the semiconductor crystal is randomized by randomly indexing into the linspace of the x and y axes. The magnitudes of all electrons' speeds are kept constant throughout the whole simulation at the thermal velocity. However, the angle is randomized to create reasonable variation throughout the simulations.

```
Px = zeros(nElectrons, nTime);
Py = zeros(nElectrons, nTime);

for n = 1 : nElectrons
    Px(n, 1) = x(randi(400));
    Py(n, 1) = y(randi(400));
end

Vx = zeros(nElectrons, nTime);
Vy = zeros(nElectrons, nTime);

for k = 1 : nElectrons
    Vx(k, :) = vth * cos(2*pi*rand());
    Vy(k, :) = vth * sin(2*pi*rand());
end
```

The following block of code updates the electrons' positions and velocities (as necessary) after each time step. This code models the simulation such that when an electron's position exceeds the x-axis limits, the electron will appear at the opposite end of the axis with the same velocity. When the electron's position reaches a y-axis limit, the electron will deflect by inverting its y-velocity.

```
for j = 1 : nElectrons
    for w = 2 : nTime
        if isnan(Px(j, w-1))
            if left == 1
                Px(j, w) = 0 + Vx(j, w-1)*dt;
            end
            if right == 1
                Px(j, w) = 200e-9 + Vx(j, w-1)*dt;
            end
        else
            Px(j, w) = Px(j, w-1) + Vx(j, w-1)*dt;
        end

        if Px(j, w) > 200e-9
            left = 1;
            right = 0;
            Px(j, w) = NaN;
        end
        if Px(j, w) < 0
            left = 0;
            right = 1;
            Px(j, w) = NaN;
        end

        Py(j, w) = Py(j, w-1) + Vy(j, w-1)*dt;
```

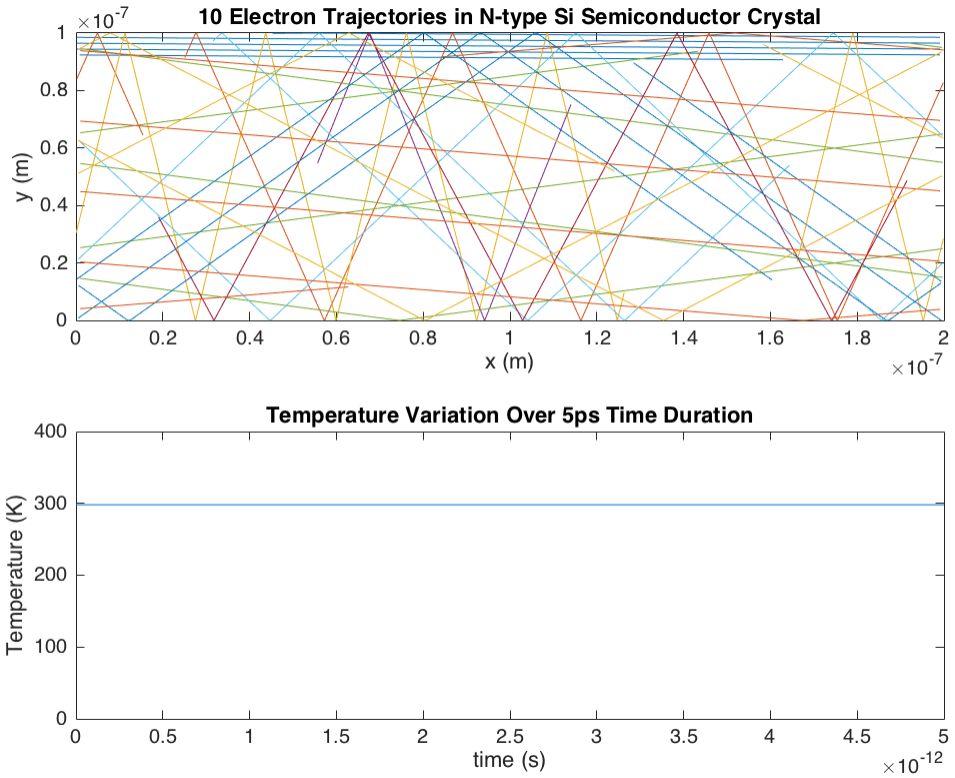
```
        if Py(j, w) > 100e-9
            Py(j, w) = 100e-9;
            Vy(j, w:end) = -Vy(j, w-1);
        end
        if Py(j, w) < 0
            Py(j, w) = 0;
            Vy(j, w:end) = -Vy(j, w-1);
        end
    end
end
```

After doing a full analysis for each particle, the temperature analysis can be performed using the following code.

```
for i = 1:nTime
        temperature(i) = (sum(Vx(:, i).^2) + sum(Vy(:, i).^2))*mn/
Kb/2/nElectrons;
        if i > 1
            time(i) = time(i-1) + dt;
        end
end
```

The following blocks of code plot the trajectory data of each electron and the total temperature of the system over a time period (5 ps).

```
figure(1)
for g = 1:10
    subplot(2, 1, 1)
    plot(Px(g, :), Py(g, :))
    xlabel('x (m)')
    ylabel('y (m)')
    title('10 Electron Trajectories in N-type Si Semiconductor
 Crystal')
    hold on
end

subplot(2, 1, 2)
plot(time, temperature)
title('Temperature Variation Over 5ps Time Duration')
xlabel('time (s)')
ylabel('Temperature (K)')
ylim([0 400])
xlim([0 5e-12])
hold off
```



Part 2: Collisions with Mean Free Path

The speeds of all of electrons will not be equal in this simulation. Instead, the speeds will be initialized using a Maxwell-Boltzmann distribution and the inital angles will be randomized. The same electron behaviour that was modelled in part 1 will be used here as well. However, this simulation introduces scattering using the following probability.

```
Pscat = 1 - exp(-dt/tau)
```

```
Pscat =

    0.0264
```

The electrons positions and velocities are initialized using the following blocks of code.

```
Px = zeros(nElectrons, nTime);
Py = zeros(nElectrons, nTime);

for n = 1 : nElectrons
    Px(n, 1) = x(randi(400));
    Py(n, 1) = y(randi(400));
end

Vx = zeros(nElectrons, nTime);
```

```
Vy = zeros(nElectrons, nTime);

MaxwellBoltzmannVdist = makedist('Normal', 'mu', 0, 'sigma',
 sqrt(Kb*T/mn));

for k = 1 : nElectrons
    Vx(k, :) = random(MaxwellBoltzmannVdist);
    Vy(k, :) = random(MaxwellBoltzmannVdist);
end
```

The average velocity should be the thermal velocity as calculated in part 1. Therefore, a simple calculation to determine the average velocity is done to make sure the distribution was set up correctly.

```
avgV = sqrt(sum(Vx(:, 1).^2)/nElectrons + sum(Vy(:, 1).^2/nElectrons))
```

```
avgV =

   1.8579e+05
```

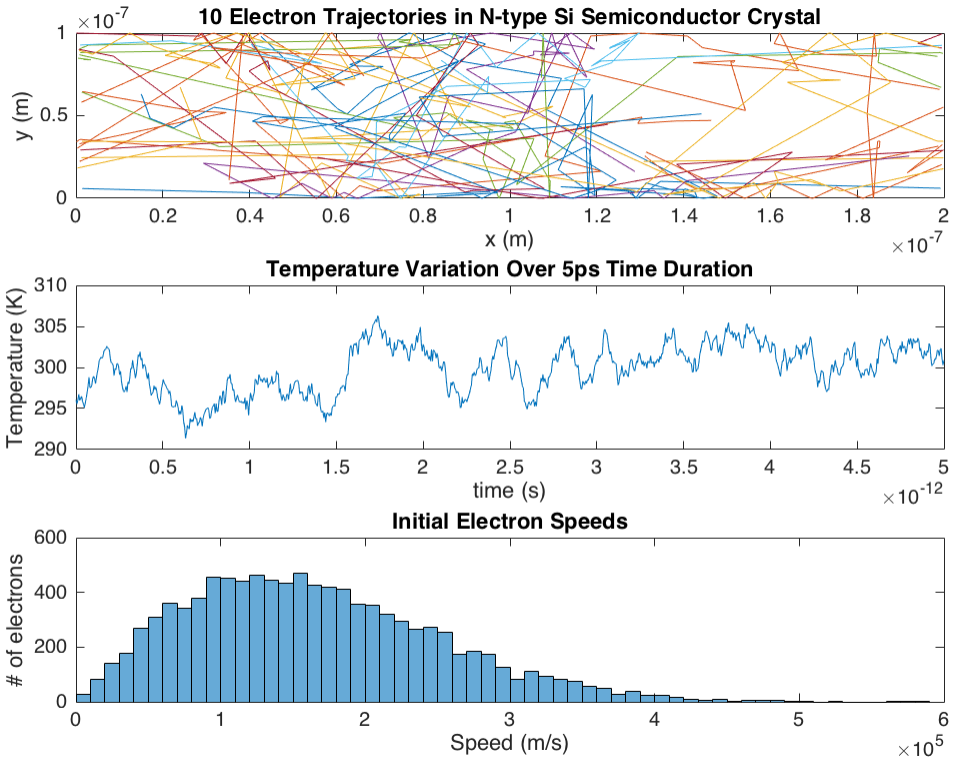
The following blocks of code perform a full analysis of the electrons' positions and velocities (updating after each time step) followed by a temperature analysis of the entire system. This block of code includes a randomized check for each electron after each time step to determine whether it scatters or not. When an electron scatters, the code assigns a new random velocity to the electron using the already generated Maxwell-Boltzmann distribution.

```
for j = 1 : nElectrons
    for w = 2 : nTime
        if isnan(Px(j, w-1))
            if left == 1
                Px(j, w) = 0 + Vx(j, w-1)*dt;
            end
            if right == 1
                Px(j, w) = 200e-9 + Vx(j, w-1)*dt;
            end
        else
            Px(j, w) = Px(j, w-1) + Vx(j, w-1)*dt;
        end

        if Px(j, w) > 200e-9
            left = 1;
            right = 0;
            Px(j, w) = NaN;
        end
        if Px(j, w) < 0
            left = 0;
            right = 1;
            Px(j, w) = NaN;
        end

        Py(j, w) = Py(j, w-1) + Vy(j, w-1)*dt;
        if Py(j, w) > 100e-9
            Py(j, w) = 100e-9;
```

```
            Vy(j, w:end) = -Vy(j, w-1);
        end
        if Py(j, w) < 0
            Py(j, w) = 0;
            Vy(j, w:end) = -Vy(j, w-1);
        end
        if Pscat > rand()
            Vx(j, w:end) = random(MaxwellBoltzmannVdist);
            Vy(j, w:end) = random(MaxwellBoltzmannVdist);
        end
    end
end

for i = 1:nTime
        temperature(i) = (sum(Vx(:, i).^2) + sum(Vy(:, i).^2))*mn/
Kb/2/nElectrons;
        if i > 1
            time(i) = time(i-1) + dt;
        end
end
```

After completing a full analysis of each electron over each time step, the data can be plotted. A histogram displaying the electron's inital velocities is also plotted to display the Maxwell-Boltzmann distribution.

```
figure(2)
for g = 1:10
    subplot(3, 1, 1)
    plot(Px(g, :), Py(g, :))
    xlabel('x (m)')
    ylabel('y (m)')
    title('10 Electron Trajectories in N-type Si Semiconductor
 Crystal')
    hold on
end

subplot(3, 1, 2)
plot(time, temperature)
title('Temperature Variation Over 5ps Time Duration')
xlabel('time (s)')
ylabel('Temperature (K)')
xlim([0 5e-12])

subplot(3, 1, 3)
vel = sqrt(Vx(:, 1).^2 + Vy(:, 1).^2);
histogram(vel)
title('Initial Electron Speeds')
xlabel('Speed (m/s)')
ylabel('# of electrons')
hold off
```



The temperature data shows that the temperature is not constant over the entire time period. The plot shows noticeable fluctuations occurring both above and below 300 K. This property of the system is due to the scattering. Also, the velocity distibution of the electrons in the system resembles a Maxwell-Boltzmann distribution.

Part 3: Enhancements

This part will introduce a bottleneck barrier to the semiconductor crystal. The electrons will not be able to pass through the barriers nor will they be initialized within the barriers' limits. The following code initializes the positions and velocities of all electrons and enforces the barriers' limits. In addition, the velocities continue to be initialized randomly from a Maxwell-Boltzmann distribution.

```
Px = zeros(nElectrons, nTime);
Py = zeros(nElectrons, nTime);

for n = 1 : nElectrons
    Px(n, 1) = x(randi(400));
    Py(n, 1) = y(randi(400));
    while (Px(n, 1) >= 80e-9 && Px(n, 1) <= 120e-9 && Py(n, 1) >=
 60e-9) || (Px(n, 1) >= 80e-9 && Px(n, 1) <= 120e-9 && Py(n, 1) <=
 40e-9)
        Px(n, 1) = x(randi(400));
        Py(n, 1) = y(randi(400));
    end
end
```

```
Vx = zeros(nElectrons, nTime);
Vy = zeros(nElectrons, nTime);

MaxwellBoltzmannVdist = makedist('Normal', 'mu', 0, 'sigma',
 sqrt(Kb*T/mn));

for k = 1 : nElectrons
    Vx(k, :) = random(MaxwellBoltzmannVdist);
    Vy(k, :) = random(MaxwellBoltzmannVdist);
end
```

The following blocks of code update the electrons' positions and velocities after each time step. The probabilistic scattering component was included again (as from Part 2) in the modelling of the electrons' behaviour. The bottleneck barrier was implemented such that when an electron comes in contact with the outer limits of the barrier, it will deflect as it does when it reaches a y-axis limit.

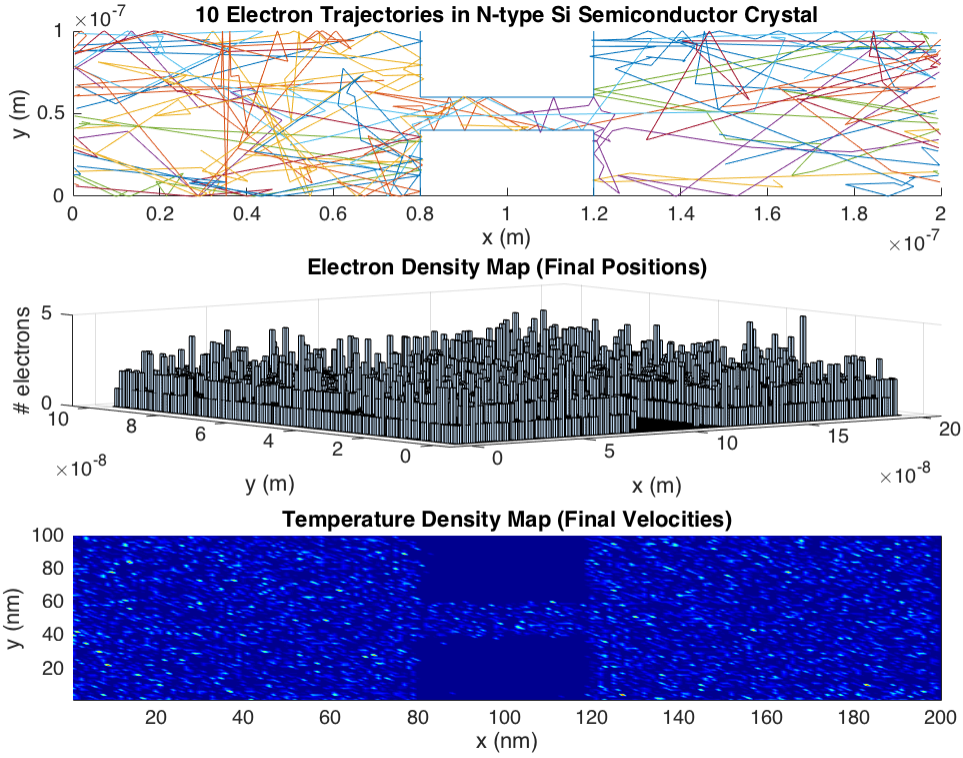
```
for j = 1 : nElectrons
    for w = 2 : nTime
        if isnan(Px(j, w-1))
            if left == 1
                Px(j, w) = 0 + Vx(j, w-1)*dt;
            end
            if right == 1
                Px(j, w) = 200e-9 + Vx(j, w-1)*dt;
            end
        else
            Px(j, w) = Px(j, w-1) + Vx(j, w-1)*dt;
        end

        if Px(j, w) > 200e-9
            left = 1;
            right = 0;
            Px(j, w) = NaN;
        end
        if Px(j, w) < 0
            left = 0;
            right = 1;
            Px(j, w) = NaN;
        end

        Py(j, w) = Py(j, w-1) + Vy(j, w-1)*dt;
        if Py(j, w) > 100e-9
            Py(j, w) = 100e-9;
            Vy(j, w:end) = -Vy(j, w-1);
        end
        if Py(j, w) < 0
            Py(j, w) = 0;
            Vy(j, w:end) = -Vy(j, w-1);
        end
        if Pscat > rand()
            Vx(j, w:end) = random(MaxwellBoltzmannVdist);
            Vy(j, w:end) = random(MaxwellBoltzmannVdist);
        end
```

```
        if (Px(j, w) >= 80e-9 && Px(j, w) <= 120e-9 && Py(j, w) >=
 60e-9) || (Px(j, w) >= 80e-9 && Px(j, w) <= 120e-9 && Py(j, w) <=
 40e-9)
            if (Px(j, w-1) <= 80e-9 && Py(j, w-1) <= 40e-9) || (Px(j,
 w-1) <= 80e-9 && Py(j, w-1) >= 60e-9) || (Px(j, w-1) >= 120e-9 &&
 Py(j, w-1) <= 40e-9) || (Px(j, w-1) >= 120e-9 && Py(j, w-1) >= 40e-9)
                Vx(j, w:end) = -Vx(j, w-1);
            end
            if Px(j, w-1) >= 80e-9 && Px(j, w-1) <= 120e-9 && Py(j,
 w-1) <= 60e-9 && Py(j, w-1) >= 40e-9
                Vy(j, w:end) = -Vy(j, w-1);
            end
        end
    end
end
```

After a full analysis is complete, the data can be plotted. Additionally, an electron density plot was included to display the arrangement of the electrons at the final time step. Also, a temperature density map was plotted to display the differences in temperature across the semiconductor crystal.

```
figure(3)
for g = 1:10
    subplot(3, 1, 1)
    xLim = [80e-9 80e-9 120e-9 120e-9];
    yLim1 = [0 40e-9 40e-9 0];
    yLim2 = [100e-9 60e-9 60e-9 100e-9];
    line(xLim, yLim1)
    hold on
    line(xLim, yLim2)
    plot(Px(g, :), Py(g, :))
    xlabel('x (m)')
    ylabel('y (m)')
    title('10 Electron Trajectories in N-type Si Semiconductor
 Crystal')
end

densityM = [Px(:, 1000) Py(:, 1000)];
subplot(3, 1, 2)
hist3(densityM, [200 100])
title('Electron Density Map (Final Positions)')
xlabel('x (m)')
ylabel('y (m)')
zlabel('# electrons')

tempx = zeros(ceil(200), ceil(100));
tempy = zeros(ceil(200), ceil(100));
tempn = zeros(ceil(200), ceil(100));

for z = 1:nElectrons
    x = floor(Px(z, 1000)/1e-9);
    y = floor(Py(z, 1000)/1e-9);
    if (x == 0 || isnan(x))
        x = 1;
    end
```

```
    if (y == 0 || isnan(y))
        y = 1;
    end
    tempy(x, y) = tempy(x, y) + Vy(z, 1000)^2;
    tempx(x, y) = tempx(x, y) + Vx(z, 1000)^2;
    tempn(x, y) = tempn(x, y) + 1;
end

temp2 = (tempx + tempy) .* mn ./ Kb ./ 2 ./ tempn;
temp2(isnan(temp2)) = 0;
temp2 = temp2';
subplot(3, 1, 3)
xtemp = linspace(1, 200, 200);
ytemp = linspace(1, 100, 100);
pcolor(xtemp, ytemp, temp2)
shading interp
colormap(jet)
title('Temperature Density Map (Final Velocities)')
xlabel('x (nm)')
ylabel('y (nm)')
hold off
```



The position data shows that the bottleneck barrier was correctly impelemented and the desired behaviour in the crystal was achieved. The temperature density map shows the bottlenack barrier as well.

Published with MATLAB® R2015b